Computer Science and Application 计算机科学与应用, 2023, 13(8), 1576-1587
Published Online August 2023 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2023.138156

Hans汉斯

基于深度学习的佩戴口罩下的人脸识别

王欣雅，林泓旭，吕尚颖，黄睿宸，聂敬儿

大连外国语大学软件学院，辽宁 大连

收稿日期：2023年7月14日；录用日期：2023年8月11日；发布日期：2023年8月18日

摘　要

深度学习卷积神经网络在图像处理中的应用引起了国内外许多学者的广泛关注。识别和验证有遮挡物下的人脸将是深度学习领域里持续受到关注的课题，我们需要更有效的方法来实现实时佩戴口罩检测和面部识别。从传统的机器学习算法到现在的深度学习卷积神经网络，图像识别效率、图像识别精度和网络训练速度的优化始终都是第一要义。为解决传统神经网络的梯度消失和网络退化问题，本文提到了一种基于改进型激活函数LeakyReLU的ResNet18残差神经网络的口罩遮挡下的人脸识别方法。利用Python语言构建PyTorch框架下的ResNet18残差神经网络模型，训练结果显示，改进型激活函数LeakyReLU在两轮训练后产生的结果比同等训练条件下ReLU函数的识别精确度高，因此，ResNet18卷积神经网络模型较其他人脸遮挡识别方法在识别准确度上有所提升。

关键词

口罩遮挡下的人脸识别，深度学习，卷积神经网络，ResNet18，LeakyReLU

Masked Face Recognition Based on Deep Learning

Xinya Wang, Hongxu Lin, Shangying Lv, Ruichen Huang, Jing'er Nie

School of Software, Dalian University of Foreign Languages, Dalian Liaoning

Received: Jul. 14th, 2023; accepted: Aug. 11th, 2023; published: Aug. 18th, 2023

Abstract

The application of deep learning convolutional neural network in image processing has attracted wide attention of many scholars at home and abroad. Recognizing and verifying faces under occlusions will continue to be a hot topic in the field of deep learning. We need more effective methods for real-time mask wearing detection and face recognition. From the traditional machine learning

文章引用: 王欣雅, 林泓旭, 吕尚颖, 黄睿宸, 聂敬儿. 基于深度学习的佩戴口罩下的人脸识别[J]. 计算机科学与应用, 2023, 13(8): 1576-1587. DOI: 10.12677/csa.2023.138156

algorithm to the current deep learning convolutional neural network, the optimization of image recognition efficiency, image recognition accuracy and network training speed is always the first essential. In order to solve the problem of gradient disappearance and network degradation of traditional neural network, this paper mentions a ResNet18 residual neural network based on improved activation function LeakyReLU for face recognition under mask occlusion. Python language is used to build a ResNet18 residual neural network model under the PyTorch framework. The training results show that the improved activation function LeakyReLU produces higher recognition accuracy than the ReLU function under the same training conditions after two rounds of training. The ResNet18 convolutional neural network model has improved the recognition accuracy compared with other face occlusion recognition methods.

Keywords

Face Recognition under Mask Occlusion, Deep Learning, Convolutional Neural Network, ResNet18, LeakyReLU

1. 绪论

1.1. 研究的目的和意义

由于 COVID-19 的余波和季节性流感的传播，人们佩戴口罩出行已经成为生活的常态。与此同时，佩戴口罩造成的人脸大面积遮挡给人脸识别技术带来了新的挑战。为了解决这个问题，口罩人脸识别(MFR)系统应运而生。口罩对于人脸识别造成的影响主要体现在其对人脸的遮挡导致了大量用于分析人脸特征的信息缺失，从而影响人脸关键点定位的准确性进而影响对人脸特征的提取[1]。同时，在人脸特征缺失的情况下，口罩本身也对识别造成了干扰，人脸的特征表达发生改变，无法与系统存储的人脸特征记录相匹配，传统的快速人脸识别技术不再有效，戴口罩的人脸识别已经成为研究人员的重要任务。

1.2. 戴口罩人脸识别国内外的研究现状

1.2.1. 国内研究现状

国内许多大型科技公司和高校都在这个领域进行了相关研究和开发，但目前公开的数据集与实验细节并不多。国内的研究主要集中在数据集的采集和算法的优化方面，采用了许多国内外经典网络结构和预训练模型进行实验，并尝试对数据集进行增广、剪枝等操作，以提高模型的准确率。但是，国内的研究大多是基于小规模的数据集进行的，缺乏大规模的口罩下人脸识别数据集，这导致该技术的应用场景受到很大的限制。

1.2.2. 国外研究现状

口罩下的人脸识别技术在全球范围都得到了广泛的关注，越来越多的国外研究学者探索使用增量学习、迁移学习、轻量化等技术优化口罩下的人脸识别性能，以提升模型的识别精度。尽管采用已有的深度学习算法和技术已经取得了不错的结果，但在处理具有大面积遮挡情况下的面部识别时，模型的准确率和鲁棒性仍需进一步提高。

2. 深度残差网络

2.1. 传统神经网络面临的窘境

深度卷积神经网络的发展给图像分类带来一系列便利，更深层的网络就意味着拥有更多的参数，更多的参数会使得模型的训练效果更好，但是这些网络都面临最显著的梯度消失和梯度爆炸问题，适当的权重初始化或使用 Batch Normalization 层可以有效缓解梯度问题。虽然学者们尝试努力解决梯度消失和爆炸问题，但新的问题接踵而至。实验数据证明，随着网络深度的增加，训练集精确度变得饱和之后模型的准确度开始迅速退化，见图 1。

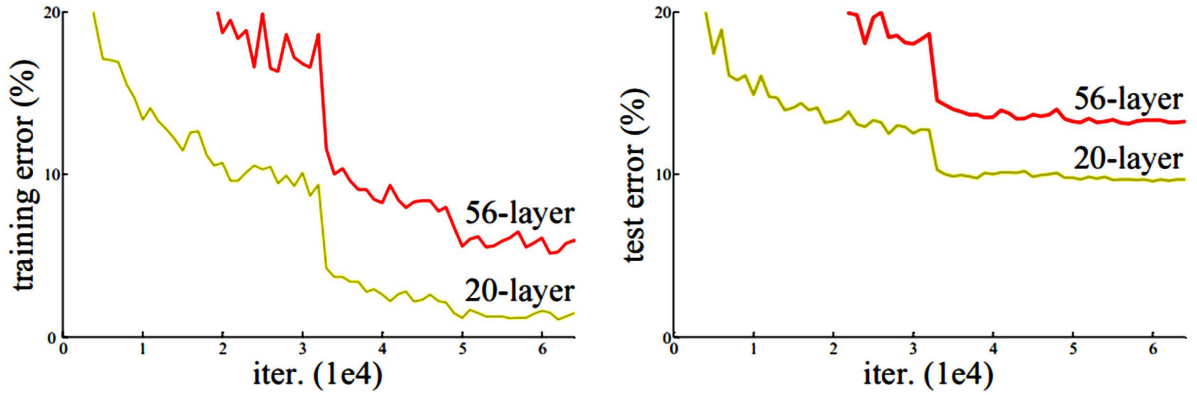

Figure 1. Deeper network model will have higher training errors and test errors (based on CIFAR-10 dataset) [2]
图 1. 更深层的网络模型会有更高的训练误差和测试误差(基于 CIFAR-10 数据集) [2]

在以往的深度学习神经网络模型中，层数一般都不超过 20 层。研究人员已经在实验中证实，通过堆加层数并不能很好地优化模型，神经网络的退化问题表明不是所有的系统都很容易优化，因此，在避免梯度爆炸的同时研究如何优化网络模型是必要的。

2.2. 创新理论下诞生的残差学习模块

我们可以从浅层网络入手来构建模型，在此基础上不断叠加层数得到更深层的网络模型。假设我们的神经网络在层数为 l 的时候打到了最优的效果，那么在 l 层的基础上去构建更深层的网络是一个恒等映射，但是在深度学习中拟合一个恒等映射是非常困难的。如果用 x 来表示目前学到的内容，用 $H(x)$表示期望得到的映射，那么拟合 $H(x)$和 x 的残差也就是优化拟合 $H(x)$-x [3]。如果我们能将 $F(x)$训练为 0，理论上可以认为我们成功地构造出了难以拟合的恒等映射。

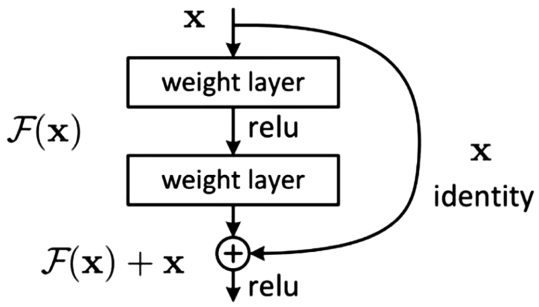

Figure 2. Residual module structure diagram [2]
图 2. 残差模块结构图[2]

如图 2 的残差模块中，输入 x 分为两个方向进行，而不是像大多神经网络中层层传递的模型。首先输入 x 在主分支经过一层权重层并 Relu 激活，第二次通过权重层后不进行激活。侧分支方向通过 shortcut 使得 x 直接和主分支处理过后的结果相加，和在经过一次 Relu 激活后传给下一层。

2.3. 残差网络模型的架构

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3. ResNet model with five layer structures
图 3. 五种层结构 ResNet 模型

ResNet 开发者给出了五种网络架构模型，见图 3，ResNet34 中涉及最大池化和平均池化操作，最大池化是指对每一个块中的数据取最大值计算，平均池化是指对每一个块的数据进行平均值运算。

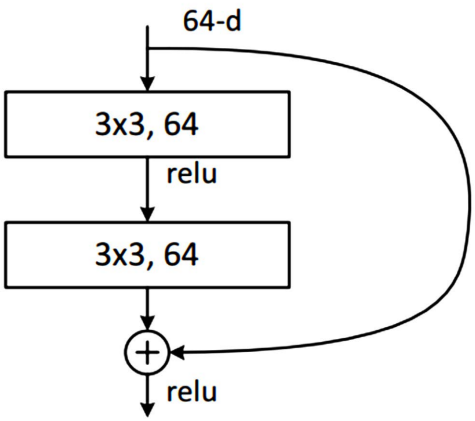

Figure 4. Internal structure of the 3*3 residual module [2]
图 4. 3*3 的残差模块内部结构[2]

通过主分支进行 3*3 卷积运算后经 Relu 函数激活，在进行 3*3 卷积运算，最后与原输入相加之后，在经过 Relu 函数激活之后输入至下一层，见图 4。对于实线标注的残差结构，输入矩阵的 shape 和输出矩阵的 shape 是一模一样的。而对于虚线标注的残差结构，由于输入矩阵和输出矩阵的 shape 不同，我们必须对步长和卷积核进行处理来完成传递。

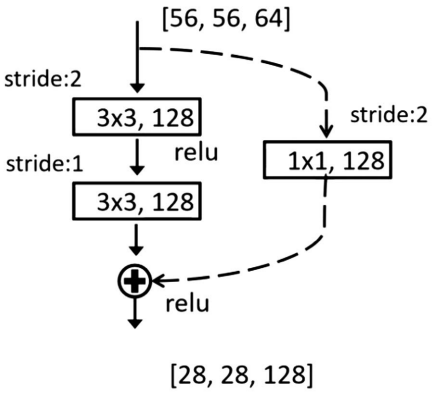

Figure 5. Internal structure of the 3*3 residual module [2]
图 5. 3*3 的残差模块内部结构[2]

对于 shape 为[56,56,64]的输入矩阵，见图 5，在主分支上，由于 stride 为 2，特征向量的高和宽缩减为原来的一半，之后经过该层 128 个卷积核的处理，使得特征矩阵的深度从 64 变为 128，保证了本层的结果能够正确的传递至下一层。

2.4. 解决梯度问题算法——批标准化

传统神经网络中，我们主要使用反向传播算法来优化深度学习神经网络，即根据损失函数计算的误差通过梯度反向传播的方式指导深度网络权值的更新优化，但梯度消失和梯度爆炸问题会随着网络层数的增加变得越来越明显，网络会变得不稳定[4]。

使用 Batch Normalization 层可以显著加速深度神经网络的训练，并有效地解决网络不稳定问题。该层通过归一化修正层输入的均值和方差，从而降低了梯度对参数或它们的初始值尺度的依赖关系，有益于通过网络的梯度流动。

2.5. 激活函数的选择

引入激活函数的主要目的是增加网络的非线性特性，从而提高网络的表达能力和拟合能力。在卷积神经网络中，卷积操作和池化操作等线性操作只能对输入数据做线性变换，而激活函数可以将线性变换后的结果进行非线性映射，从而使网络能够更好地处理复杂的非线性关系。不引入激活函数的神经网络可能会退化为感知机模型，使它无法准确处理线性不可分数据。常用的激活函数包括 Sigmoid、tanh、ReLU、softmax 等。不同的激活函数具有不同的性质和适用场景，需要根据具体任务和网络结构选择合适的激活函数。

2.5.1. Sigmoid 函数

Sigmoid 函数公式：$f(x)=\dfrac{1}{1+e^{-x}}$

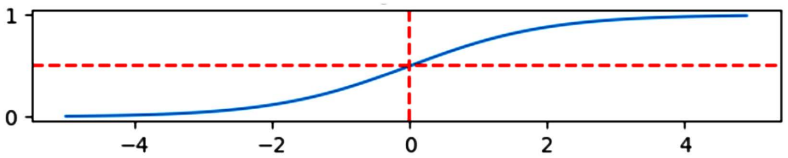

Figure 6. Image of Sigmod function
图 6. Sigmod 函数图像

如图 6 函数图像可以看出，Sigmoid 函数可以将输入的连续数值变换为 0 到 1 之间的值，相当于进行数值压缩。当遇到较大或较小的输入值时，计算的梯度趋于 0。反向过程中再进行链式法则叠乘就会造成梯度消失现象[5]。

2.5.2. tanh 函数

tanh 函数公式： $\tanh x = \dfrac{\sinh x}{\cosh x} = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

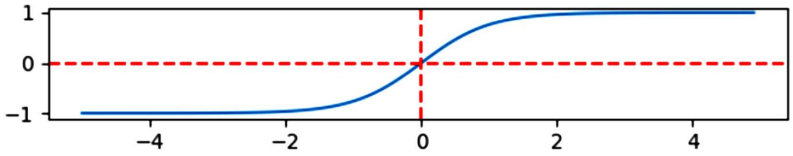

Figure 7. Image of the tanhx function
图 7. tanhx 函数图像

相比 Sigmoid 函数，tanh 激活函数的输出值在正负区间都有分布，但是仍然没有解决梯度消失问题。x 的绝对值越大，其函数值的绝对值越接近 1 且越平滑，梯度消失问题依然会出现[6]，见图 7。

2.5.3. ReLU 函数

ReLU 函数表达式：ReLU = max(0, x)

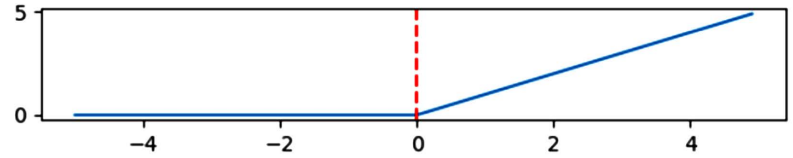

Figure 8. Image of ReLU function
图 8. ReLU 函数图像

当输入的数据为负数时，神经元不会被激活，神经网络只会将大于 0 的数据输出并激活，极大地提高了计算效率。由此可见，ReLU 函数在快速收敛的同时还可以缓解梯度消失并提高计算速度，见图 8。

3. Residual Network 的新理解与改进

3.1. 从集成学习角度理解 ResNet

若从数学的角度解释，可以假设三个残差块模型 f_1、f_2、f_3，输入的数据为 x_0，由于残差网络具有递归性质，我们可以得出递推关系式： $x_{i+1} = f_i(x_i) + x_i$ 。

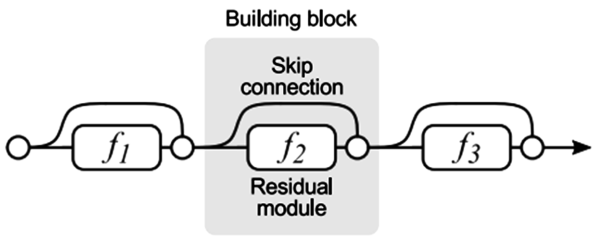

Figure 9. Visualization of three residual blocks [7]
图 9. 三个残差块可视化[7]

展开递推关系式：

$$\begin{aligned} x_3 &= x_2 + f_3(x_2) \\ &= x_1 + f_2(x_1) + f_3(x_1 + f_2(x_1)) \\ &= x_0 + f_1(x_0) + f_2(x_0 + f_1(x_0)) + f_3\left[x_0 + f_1(x_0) + f_2(x_0 + f_1(x_0))\right] \end{aligned}$$

通过以上表达式，我们可以将图 9 形象化为以下结构：

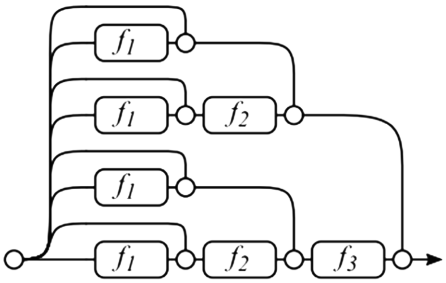

Figure 10. Visualization of conventional three residual blocks [7]
图 10. 常规三残差块形象化视图[7]

从见图 10 我们可以发现，每一个模块都有两种路径选择：第一种选择跳过连接，第二种选择通过残差模块，对于 n 块残差模块，进行形象化视图后能得到 2^n 条路径。通过上述分析，我们可以得知残差网络的路径并不是相互依赖的。

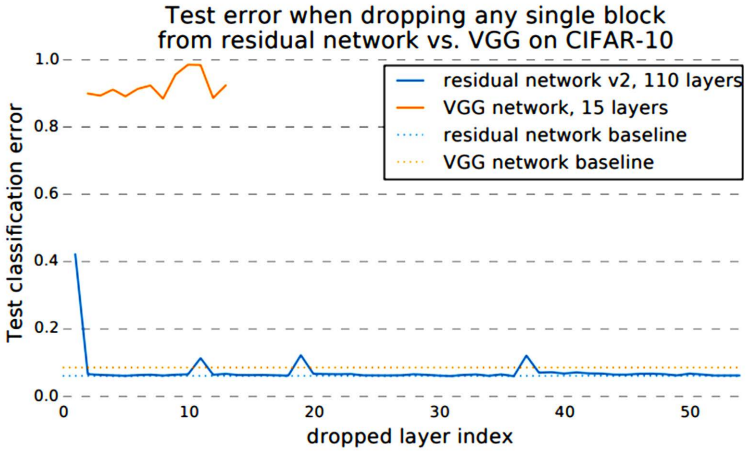

Figure 11. Comparison experiment of deleting a single layer [7]
图 11. 删除单个层的对比实验[7]

图 11 表明，对于 VGGNet，删除单个层会造成整个网络的性能降低至随机级别，而对于 ResNet 网络，删除某一层还会保留一半的路径，实验结果也表明删除某一层也能使得整个网络的训练效果不会发生较大幅度的变化。在训练过程中，由于每个残差模块都可以独立地进行训练，因此可以将 ResNet 看作一种集成学习的方法，将多个浅层模型集成一个深层模型，有效提高了网络的表达能力。

3.2. 改进型激活函数

ReLU 激活函数被普遍应用于神经网络中，但在 ReLU 函数中，负数的输入会被激活函数映射为 0，

这将会导致某些神经元可能永远不会被激活，后续学习参数将不再更新，这种现象可以称为 DEAD ReLU 问题。为了解决 DEAD ReLU 问题，我们可以引进 PReLU 函数对 ReLU 函数进行一定幅度的改进[8]。

$$\mathrm{PReLU}\left(x_i\right)=\begin{cases}x_i & \text{if } x_i>0\\ a_i x_i & \text{if } x_i\le 0\end{cases}$$

相比于 ReLU 函数，PReLU 函数并不是直接将负值变为 0，而是在负值区间内引进了系数 α，这可以使得输入在负值区域内的数据能以微小的速度进行更新迭代。如果令系数 $\alpha=0.1$，则称该函数为 Leaky ReLU 函数，如图 12 所示。

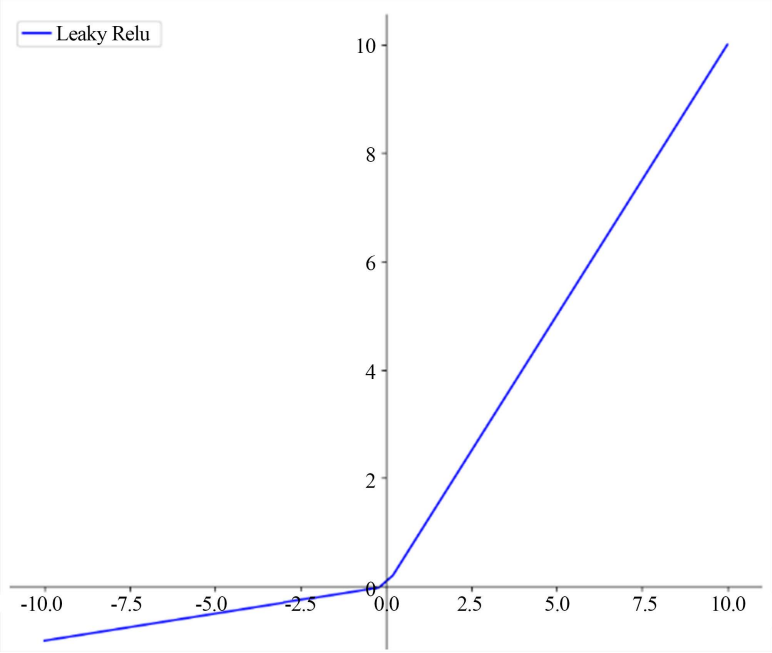

Figure 12. Leaky ReLU function image
图 12. Leaky ReLU 函数图像

改进后的 Leaky ReLU 函数不仅具有 ReLU 函数的优点，还能解决参数停止更新的问题，使得 ResNet 模型的效果更加优秀。

4. PyTorch 框架下的 ResNet 人脸口罩遮挡识别网络搭建

4.1. PyTorch 简介

由于 Torch 开发语言并未得到广泛的应用，开发团队决定使用 Python 语言进行全面重构，PyTorch 由此诞生。一经推出，PyTorch 就在深度学习领域得到了广泛应用，特别是在计算机视觉与自然语言处理等领域。PyTorch 提供 CPU 设备和 GPU 设备的支持和加速，能有效提高数据运行速度。另外，Pytorch 也支持动态神经网络，其库函数中包含了大量已经封装完成的深度学习神经网络框架，为模型的构建和项目的研究提供了便利。

4.2. 数据集构建

将 8000 张公开的图像数据及网络上的人脸图片(部分图片实例见图 13)分为佩戴口罩的和未佩戴口罩两类，利用 LabelImg 进行标注并进行数据预处理。

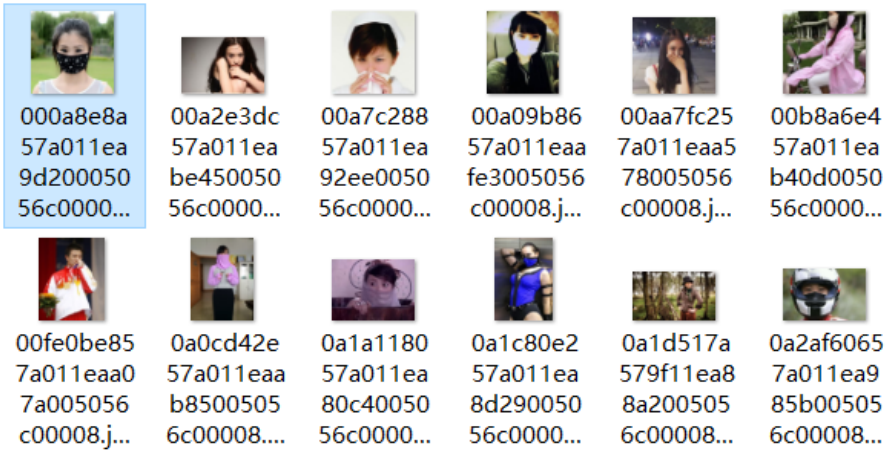

Figure 13. Part of the image
图 13. 部分图像

4.3. ResNet 网络搭建

本实验基于 PyCharm 完成，实验所用框架为 ResNet18 框架，见图 14。

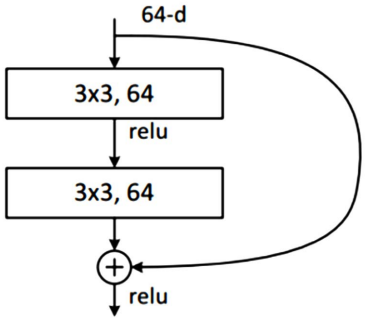

Figure 14. Residual structure of ResNet18 [2]
图 14. ResNet18 的残差结构[2]

4.3.1. 残差结构块代码实现

```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_channel, out_channel, stride=1, downsample=None, **kwargs):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channel, out_channels=out_channel,
                               kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=out_channel, out_channels=out_channel,
                               kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        self.downsample = downsample
```

Figure 15. Residual structure block code implementation
图 15. 残差结构块代码实现

图 15 中参数含义：

expansion：对应残差结构中主分支卷积核是否发生变化的参数

in_channel&out_channel：输入和输出特征矩阵的深度

stride：步长

downsample：下采样参数，默认为 None

其中下采样参数对应前文图 2、图 5 提出的残差网络中虚线和实线网络结构问题，设置该参数通过判断来完成降维工作。按照残差结构的顺序，构建 conv1 层并通过 bn 层由激活函数激活，传入 bn 层的参数是经过 conv1 层之后的矩阵深度，之后构建 conv2 层并通过 bn 层。在 conv1 和 conv2 中的 stride 设置不同，是因为每层的虚线结构要对矩阵的高和宽进行缩减一半的操作。

4.3.2. 传参并调用 ResNet 函数

```
def resnet18(num_classes=1000, include_top=True):

    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes, include_top=include_top)
```

Figure 16. Passing parameters and calling ResNet function
图 16. 传参并调用 ResNet 函数

调用 ResNet18 函数，传入已经定义完成的 BasicBlock 类，残差层个数为[2,2,2,2]，最后输入类别个数即可完成调用，见图 16。

4.4. 训练网络与实验验证

图片

进行数据预处理并构建 ResNet 网络架构之后，我们编译主函数构建训练过程，分别利用公有测试集和私有测试集进行验证。由于本实验使用 CPU 进行网络训练，训练速度较慢，所以仅进行两个 epoch 的学习并验证。

训练结果如图 17。

```
Epoch: 0
learning_rate: 0.01
 [========== 14355/14355 =====>] | Loss: 1.963 | Acc: 25.668% (7369/28709)
mainpro_FER.py:74: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  inputs, targets = Variable(inputs, volatile=True), Variable(targets)
 [========== 1795/1795 =======>] | Loss: 1.870 | Acc: 30.844% (1107/3589)
Saving..
best_PublicTest_acc: 30.844
mainpro_FER.py:115: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  inputs, targets = Variable(inputs, volatile=True), Variable(targets)
 [========== 1795/1795 =======>] | Loss: 1.876 | Acc: 30.900% (1109/3589)
Saving..
best_PrivateTest_acc: 30.900

Epoch: 1
learning_rate: 0.01
 [========== 14355/14355 =====>] | Loss: 1.779 | Acc: 33.467% (9608/28709)
 [========== 1795/1795 =======>] | Loss: 1.558 | Acc: 39.287% (1410/3589)
Saving..
best_PublicTest_acc: 39.287
 [========== 1795/1795 =======>] | Loss: 1.559 | Acc: 39.900% (1432/3589)
Saving..
best_PrivateTest_acc: 39.900
best_PublicTest_acc: 39.287
best_PublicTest_acc_epoch: 1
best_PrivateTest_acc: 39.900
best_PrivateTest_acc_epoch: 1
```

Figure 17. ReLU function ResNet18 network training results
图 17. ReLU 函数 ResNet18 网络训练结果

如图 17 所示，对于 ResNet18，进行一个 epoch 共计 14355 的样本训练之后，在公开测试集上验证的准确率为 30.844%，私有测试集的准确率为 30.900%。进行训练两个 epoch 之后，在公开测试集上验证的准确率为 39.287%，私有测试集的准确率为 39.900%。

接下来更改为 LeakyRelu 函数并进行实验，在前向传播过程中直接调用 LeakyRelu 函数，得到实验结果如图 18 所示。

```
(pytorchDeeplearning) G:\L1990_final>python mainpro_FER.py --model Resnet18 --bs 2 --lr 0.01
==> Preparing data..
==> Building model..

Epoch: 0
learning_rate: 0.01
 [=========== 14355/14355 =====>] | Loss: 1.965 | Acc: 26.267% (7541/28709)
mainpro_FER.py:74: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  inputs, targets = Variable(inputs, volatile=True), Variable(targets)
 [=========== 1795/1795 =======>] | Loss: 1.650 | Acc: 34.188% (1227/3589)
Saving..
best_PublicTest_acc: 34.188
mainpro_FER.py:115: UserWarning: volatile was removed and now has no effect. Use `with torch.no_grad():` instead.
  inputs, targets = Variable(inputs, volatile=True), Variable(targets)
 [=========== 1795/1795 =======>] | Loss: 1.658 | Acc: 34.244% (1229/3589)
Saving..
best_PrivateTest_acc: 34.244

Epoch: 1
learning_rate: 0.01
 [=========== 14355/14355 =====>] | Loss: 1.812 | Acc: 35.452% (10178/28709)
 [=========== 1795/1795 =======>] | Loss: 1.680 | Acc: 43.940% (1577/3589)
Saving..
best_PublicTest_acc: 43.940
 [=========== 1795/1795 =======>] | Loss: 1.727 | Acc: 43.076% (1546/3589)
Saving..
best_PrivateTest_acc: 43.076
best_PublicTest_acc: 43.940
best_PublicTest_acc_epoch: 1
best_PrivateTest_acc: 43.076
best_PrivateTest_acc_epoch: 1
```

Figure 18. LeakyReLU function ResNet18 network training result
图 18. LeakyReLU 函数 ResNet18 网络训练结果

对于使用 LeakyReLU 激活函数的 ResNet18，进行一个 epoch 共计 14,355 的样本训练之后，在公开测试集上验证的准确率为 34.188%，私有测试集的准确率为 34.244%。进行两个 epoch 训练之后，在公开测试集上验证的准确率为 43.940%，私有测试集的准确率为 43.076%。

5. 结论

由于训练集比较庞大(一组训练集共计 14,355 个训练数据)，完成一个 epoch 的训练所消耗的时间在 1h 左右，公开测试集和私有测试集的测试时间也在 20 min 左右。相较于经过充分训练的 ResNet 模型，本实验受硬件条件的限制，模型的训练还远远不够，精确度也没能达到较高水准。但无论是进行一个 epoch 训练还是两个 epoch 训练，使用 LeakyReLU 函数完成的实验的精确度的确比使用 ReLU 函数的精确度要高出 3%左右，这说明改进型激活函数 LeakyReLU 存在对 ResNet 网络进行一定程度优化的可能性。

6. 口罩佩戴检测系统运行效果

用户界面有人脸录入、更新数据库和实时检测三大主要功能：人脸录入功能支持从数据库中选取图像数据和打开摄像头拍照录入两种方式。在更新数据库之后，打开摄像头进行实时检测，可以直观地看到系统不断地提示镜头中的人是谁以及她是否佩戴口罩，见图 19。

Figure 19. System operation effect diagram
图 19. 系统运行效果图

致　谢

感谢大连外国语大学软件学院及创新创业学院对本次研究的支持。

基金项目

大连外国语大学 2023 年大学生创新创业项目。

参考文献

[1] 陈奕文. 基于形态模型关键点的戴口罩人脸识别方法研究[D]: [硕士学位论文]. 重庆: 重庆理工大学, 2022.

[2] Li, S.Z. and Jain, A.K. (2011) Handbook of Face Recognition. Springer, New York.

[3] 刘嘉. 基于深度学习的图像知识提取技术研究与应用[D]: [硕士学位论文]. 成都: 电子科技大学, 2019.

[4] 张圆圆. 基于深度学习的人类表情识别方法研究[D]: [硕士学位论文]. 长春: 吉林大学, 2020.

[5] 吴思. 基于注意力网络的人群计数方法研究[D]: [硕士学位论文]. 杭州: 杭州电子科技大学, 2021.

[6] 王元辰. 基于 U-Net 网络的低光照图像增强算法的研究与实现[D]: [硕士学位论文]. 哈尔滨: 黑龙江大学, 2020.

[7] Kanaka, D.B. and Rajesh, V. (2022) A ResNet Deep Learning Based Facial Recognition Design for Future Multimedia Applications. *Computers and Electrical Engineering*, **104**, Article 108384. https://doi.org/10.1016/j.compeleceng.2022.108384

[8] Wang, Z.W., Zhang, Y.J., Pan, C.C. and Cui, Z.W. (2022) MMPCANet: An Improved PCANet for Occluded Face Recognition. *Applied Sciences*, **12**, 3144. https://doi.org/10.3390/app12063144